Worm.Win32.Zhelatin.pk Reverse Engineering

Author: Giuseppe 'Evilcry' Bonfa'
E-Mail: evilcry {AT} gmail {DOT} com
Website: http://evilcry.altervista.org
http://evilcodecave.wordpress.com

Informations about the Malware

Filename: happy-2008.exe
MD5: 0aa965b068625e8344f839c1ddc4a299
Packer: -

The Analysis

happy-2008.exe is a classical **E-Card Malware** spreaded through fake mails. The Executable gets the Current System Directory and next sets up as working directory /system32.

Next with GetFullPathNameA retrives "**C:\WINDOWS\System32\init_sys.config**", if file exists it attempts to determine its attributes, else creates a file

```
0040126A  PUSH EBX          ; /hTemplateFile => NULL
0040126B  PUSH 80           ; |Attributes = NORMAL
00401270  PUSH 2            ; |Mode = CREATE_ALWAYS
00401272  PUSH EBX          ; |pSecurity => NULL
00401273  PUSH 7            ; |ShareMode = FILE_SHARE_READ|
FILE_SHARE_WRITE
00401275  PUSH 40000000     ; |Access = GENERIC_WRITE
0040127A  LEA EAX,DWORD PTR SS:[EBP-114] ; |
00401280  PUSH EAX ; |FileName =
"C:\WINDOWS\System32\init_sys.config"
00401281  CALL DWORD PTR DS:[<&KERNEL32.CreateFile>; \CreateFileA
00401293  PUSH ESI ;Points to an Embedded Executable
00401294  PUSH EDI
00401295  MOV EDI,DWORD PTR DS:[<&KERNEL32.WriteFile>
0040129B  PUSH 0
0040129D  LEA EAX,DWORD PTR SS:[EBP-C] ;System Path
004012A0  PUSH EAX
004012A1  LEA ESI,DWORD PTR DS:[EBX+422A98] ; [config] String
004012A7  PUSH DWORD PTR DS:[ESI]
```

A file "**init_sys.config**" is created and filled with three entries:

```
[config]
[local]
[peers]
```

Successively, a series of values are attached into this config file, immediately after **[peers]** and have this form:

```
00003D6C8F338A3FDD3DF3648666F55C=0CCFC042170F00
```

and this is the piece of code after "**init_sys.config**"

```
0040132D  CALL happy-20.0040122D ;Builds init_sys.config and fill
```

```
it
00401332  LEA ECX,DWORD PTR SS:[EBP-8]
00401335  CALL happy-20.004016E8
...
00401351  CALL happy-20.00401634 ;EAX = String obtained from GetSystemTime Output
...
```

After some calls, EAX points to a new string "**init_1a30-12f1**"

```
00401391   PUSH EAX                  ; /pFilenameInPath
00401392   PUSH DWORD PTR SS:[EBP-8]; |Path
00401395   PUSH EBX                  ; |MaxPathSize
00401396   PUSH DWORD PTR SS:[EBP-4]; |FileName
00401399   CALL DWORD PTR DS:[<&KERNEL32.GetFullPat>; \GetFullPathNameA
0040139F   PUSH happy-20.004020D4     ;  ASCII ".sys"
004013A4   LEA ECX,DWORD PTR SS:[EBP-8]
004013A7   CALL happy-20.00401108
```

Inside **call 00401108** a new string is assembled "**init_1a30-12f1.sys**" please note that the numerical part of the Sys file changes at every run because it depends from GetSystemTime output.

```
004013B1   PUSH ESI
004013B2   PUSH ESI ;NULL
004013B3   CALL OpenSCManagerA
004013B9   CMP EAX,ESI
004013BB   MOV DWORD PTR SS:[EBP-C],EAX
004013BE   JE happy-20.004014D9
```

After opening **ServiceManager** for **LocalHost**, Service Status is enumerated and:

```
00401407  PUSH DWORD PTR SS:[EBP-18]   ; /Arg3
0040140A  PUSH EDI                     ; |Arg2
0040140B  PUSH DWORD PTR DS:[EBX]      ; |Arg1 = 0012FE62 ASCII "Abiosdsk"
0040140D  CALL happy-20.00401579       ; \happy-20.00401579
```

This Call compares the Services Name (abp480n5,ACPI,adpu16, etc..) present in the system with 'init_' string.

After this check a GetLastError is called:

```
0040142E  JNZ SHORT happy-20.0040143D
00401430  CALL GetLastError
00401436  CMP EAX,0EA
0040143B  JE SHORT happy-20.004013D1
```

If the Service exists and is running, the task of **happy_2008** ends here, else, a copy of a **Device Driver** is extracted from the executable and runned as **System Service.**

I've extracted that device driver with an HexEditor, it starts at **00403018** and ends at **00424FF8.**

The Driver Part

First traces can be seen into Registers (as for every Service)

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\init_xxxx-xxx]
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services\init_xxxx-xxx]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\init_xxxx-xxx]

DisplayName is the effective name of the Service, the phisical Driver Executable is hidden in **\??\C:\WINDOWS\System32\init_1056-4270.sys** so the .sys file is phisically **invisible**.

Now we will go to disassemble the Rootkit, the copy of SYS file that I've exctracted is not packed, but a friend of mine, ZaiRoN, signaled me that exists also packed versions of the driver.

In a first time is checked the **NtBuildNumber**, and if different from **3790 (Windows 2003) jumps out**, the device is created with the name **"\\Device\\DRV_MODULE_DRV"** and **SymbolicLinkName "\\DosDevices\\DRV_MODULE_DRV"** and next by using **PsCreateSystemThread** is created a **MultiThreaded** structure. The most rapid way to localize the MultiThread routines is to watch the **StartRoutine** parameter that represents the entry point for a driver thread.

1. **StartRoutine: 00010526**
2. **StartRoutine: 00010EF2**

[FirstThread]

```
00010532  push    offset SourceString ;
"\\BaseNamedObjects\\unrjeuurut"
00010537  lea     eax, [ebp+EventName]
0001053A  push    eax   ; DestinationString
0001053B  mov     dword ptr [ebp+Timeout], 0FD050F80h
00010542  call    ds:RtlInitUnicodeString
00010548  lea     eax, [ebp+Handle]
0001054B  push    eax                 ; EventHandle
0001054C  lea     eax, [ebp+EventName]
0001054F  push    eax                 ; EventName
00010550  call    ds:IoCreateNotificationEvent
```

Creates a notification event called **\\BaseNamedObjects\\unrjeuurut**

```
00010566    call    sub_106B0
0001056B    call    sub_10BB4
00010570    call    sub_108BC
```

Inside **call sub_106B0**, **Memory Write Protection** is toggled by using

```
push eax
mov eax, CR0
and eax, 0FFFEFFFFh
mov CR0, eax
pop eax
```

In the other calls, System Service Dispatch Table (SSDT) is hooked, and various routines are attached as SSDT Entries.

The most intersting procedure is accomplished in **sub_10C08(wchar_t *,int)** placed at **00010C08**, where is retrieved by using **PsLookupThreadByThreadId** thread ID relative to
"Services.exe"

After locating Thread ID, **PsLookupProcessByProcessId** is used to find PID of Services.exe, and finally PID is passed to KeAttachProcess() so the Rootkit can execute its code in the Context of Service.exe.